1. Bölüm

Spring'e Giriş

Spring Java dünyasında yazılım geliştirmeyi basitleştirmek için geliştirilmiş bir yazılım çatısıdır (framework). Spring'i diğer çatılardan ayıran en büyük özellik temellerinin dependency injection, yani bağımlılıkların enjekte edilmesi prensibine ve AOP'ye (Aspect Oriented Programming) dayanmasıdır. Kitabın ikinci bölümünde dependency injection ve dokuzuncu bölümünde AOP konusunu detaylı olarak inceleyeceğiz.

İki binli yılların başlarında kullanıma sunulan J2EE (Java Enterprise Edition) ve EJB (Java Enterprise Bean) teknolojileri Java ile kurumsal bazlı uygulamaların geliştirilmesini amaçlamaktaydı. Nitekim çok kısa bir zamanda Java kurumsal projelerde kullanılan popüler bir teknoloji haline geldi. Bunun başlıca sebeplerinden birisi de IBM gibi büyük firmaların Java'ya verdiği destektir.

Java 1996 yılında ilk sürümü ile sadece nesneye yönelik bir programla diliyken, J2EE ve EJB teknolojileri ile birlikte büyük bir açık kaynaklı yazılım camiasını da (open source community) kapsayan bir teknoloji platformu haline geldi. Büyük bir yazılım ekosistemine sahip olan Java platformu günümüzde de kurumsal projelerde kullanılan en popüler teknoloji platformudur.

Aklınıza bu kadar geniş bir kapsama alanına sahip bir teknolojinin yanında nasıl olur da Spring gibi popüler bir çatı var olabilir sorusu gelebilir. Öncelikle şunu belirtelim: JEE (Java Enterprise Edition) Spring, Spring'de JEE değildir. Her ikisi de Java dilini kullanarak uygulama geliştirmek için geliştirilmiş teknoloji platformlarıdır. Spring var olma nedenini J2EE'ye borçlu. Bunu açıklayalım.

İki binli yılların başlarında EJB 1.x ve 2.x teknolojileri ile yazılım geliştirmiş olanlar çok iyi bilirler. Bu teknolojileri kullanarak yazılım yazmak kadar programcı için zahmetli bir uğraşı yoktur. EJB yazılım geliştirme modeli bir uygulama sunucusunun kullanımını gerektirmektedir. Geliştirilen EJB uygulamalarının çalışabilmeleri için JBoss, Weblogic ya da Glassfish gibi uygulama sunucusuna ihtiyaç duyulmaktadır. Bu ilk bakışta kötü bir şey değil. Uygulama sunucuları EJB uygulamaları için ihtiyaç duydukları transaksiyon yönetimi ya da güvenlik gibi her türlü servisi sunmaktadırlar. Programcı için bu servisleri bedel ödemeden almak, onun işletme mantığına konsantre olmasını sağlamaktadır. Ama uygulama sunucusundan bu hizmetleri alabilmek için işletme mantığının da belli bir yapıda kodlanmış ve konfigüre edilmiş olması gerekmektedir. Bu en azından EJB 3 öncesi programcıların başını ağrıtan bir durum idi. Bunun yanı sıra EJB modüllerini uygulama sunucusu dışında test

etmek mümkün değildi. Ben o tarihlerde çalıştığım kurumsal projelerde çok iyi hatırlıyorum. Günün hatırı sayılır bir bölümünü uygulamayı test edebilmek için uygulama sunucularını çalıştırıp, durdurmakla geçirirdim. Uygulama sunucusu çalışmıyorsa, işletme mantığını test etmek imkansızdı. Bir EJB 2 uygulamasını uygulama sunucusu içinde tam çalışır hale getirmek en kötü şartlarda on ya da on beş dakika sürebilir. Artık gerisini siz düşünün.

EJB 2 teknolojisi sağladığı imkanlarla kurumsal gereksinimlerin hakkını verebilecek uygulamalar geliştirmeyi mümkün kılsa da, hantallığından dolayı birçok programcının tabiri caizse nefret ettiği bir teknoloji olarak tarihe geçmiştir. Bu Spring çatışının ortaya çıkmasına sebep olmuştur.

Rod Johnson 2002 yılında kaleme aldığı **Expert One-on-One: J2EE Design and Development** isimli kitabında Interface 21 adını taşıyan altyapı ile nasıl daha hızlı ve kolay kurumsal projelerin geliştirilebileceğini anlatıyor. Interface 21 daha sonra açık kaynaklı yazılıma dönüştürüldü ve bugün tanıdığımız Spring çatısının temellerini oluşturdu.

Rod Johnson'un yazılım yaparken önerdiği temel prensip EJB'ler yerine sade Java nesnelerin (POJO; Plain Old Java Object) kullanımı. Buradan yola çıkarak Spring için şunu söyleyebiliriz: Zaten yapıları itibariyle karmaşık olan kurumsal projeler EJB 2 gibi teknolojiler kullanıldığında daha da karmaşık hale gelmektedir. Spring sunduğu POJO tabanlı programlama modeli ile kurumsal projelerin daha hızlı gerçekleştirilmelerini ve yazılımcıların verimliliğini artırmayı hedeflemektedir. **Kısaca Spring Java ile yapılan yazılımı basitleştirmek ve sadeleştirmek için vardır.**

Spring Filozofisi

Özellikle nesneye yönelik programlama teknikleri kullanıldığında, nesneler arasında var alan bağımlılıklar çok karmaşık bir yapının oluşmasına neden olabilmektedir. Uygulama geliştirme esnasında bağımlılıkların kontrol altına alınmasına dair bir çalışma yapılmadığı taktirde, yazılımcının verimliliği ve uygulamanın kod kalitesi düşecektir. Kaliteyi artırmanın ve yazılımcının daha verimli olmasını sağlamanın bir yöntemi, tüm bağımlılıkların ve oluşan karmaşık yapının dış bir uygulama çatışı (framework) tarafından yönetilmesini sağlamak olabilir. Bu bağımlılıkların uygulama tarafından değil, kullanılan uygulama çatışı tarafından yönetilmesi anlamana gelmektir. Bu yazılım

filozofisine kontrolün tersine çevrilmesi ya da **Inversion of Control (IoC)** ismi verilmektedir. Spring çatısının var oluşu ve çalışma prensipleri bu filozofiye dayanmaktadır.

Dependency Injection

Java gibi nesneye yönelik bir programlama dili ile geliştirilen uygulamalar ideal şartlarda kodun tekrar kullanıldığı modüler bir yapıdadır. Modüller birbirlerini kullanarak, yapmaları gereken işlemleri gerçekleştirirler. Bu modüller arası bağımlılıkların oluşmasını sağlar.

```
class RentalController {
    private RentalService service = new RentalServiceImpl();
}
```

Yukarıda yer alan RentalController sınıfı/modülü bunun güzel bir örneğini teşkil etmektedir. RentalController sınıfı RentalService interface sınıfına bağımlıdır. Böyle bir bağımlılık modüler bir yapının oluşturulması ve kodun tekrar kullanımını sağlamak açısından zaruridir. Lakin RentalController bünyesinde somut bir implementasyon sınıfı olan RentalServiceImpl sınıfından new operatörü ile yeni bir nesne oluşturulması, mevcut bağımlılığın değiştirilemez ve tek bir tipte olması gerektiği anlamına gelmektedir. Bağımlılığı yeniden yapılandırabilmek için kodu değiştirmek ve yeniden derlemek gerekmektedir. Bağımlılıklarını kendisi yöneten bir uygulamada kod kalitesini düşüren bu tür bağımlılıkların oluşturulmasıdır. Oysaki bağımlılıkların tersine çevrilmesi prensibine (DIP; Dependendy Inversion Principle) göre bağımlığın yönü somut değil, soyut sınıflara doğru olmalıdır.

```
class RentalController {
    private RentalService service = rentalServiceFactory.instance();
}
```

Somut bir sınıfa olan bağımlılığı yok etmek için rentalServiceFactory gibi bir fabrika (factory) sınıfından faydalanabiliriz. Fabrika tasarım şablonunu simgeleyen rentalServiceFactory bünyesinde hangi somut RentalService implementasyonunun kullanıldığını gizlemekte ve RentalController sınıfını bahsettiğim somut bağımlılıktan kurtarmaktadır. Lakin buradaki sorun rentalServiceFactory nesnesine olan bağımlılıktır. Bu nesnenin de bir şekilde new operatörü ile oluşturulması gerekmektedir.

```
class RentalController {

    private RentalService service;

    public void setService(RentalService service){
        this.service = service;
    }
}
```

Bağımlılıkları oluşturma işlemi ile hiç uğraşmasak, bunu başka birisi bizim için yapsa nasıl olurdu? Yukarıda yer alan kod örneğinde service değişkenine gerekli değer setService() metodu aracılığı atanmaktadır. Biran için setService() metodunun dış bir mekanizma tarafından koşturulduğunu düşünelim. Bu mekanizma setService() metodunu kullanarak herhangi bir RentalService implementasyonunu RentalController sınıfına enjekte edebilir. Bu işleme bağımlılıkların enjekte edilmesi yani dependency injection (DI) ismi verilmektedir. Bu işlemi yapan da Spring çatışıdır.

Aşağıda tipik bir Spring XML konfigürasyon örneği yer almaktadır. Yönetimi Spring'e devredilen bağımlılıklar için bu tarz konfigürasyon dosyaları oluşturulur. Spring bu konfigürayon dosyalarını kullanarak nesnelere arası gerekli bağımlılıkları oluşturur, yani bağımlılıkları enjekte eder.

```
<bean id="rentalController"
        class="com.kurumsaljava.spring.RentalController>
    <property name="service" ref="rentalService"/>
</bean>

<bean id="rentalService"
        class="com.kurumsaljava.spring.RentalServiceImpl/>
```

Bağımlılıkların enjekte edilmesi prensibi ile çok sade yapıda olan sınıflar oluşturabiliriz. Kendi bağımlılıklarını yönetmek zorunda olmayan bir sınıf asıl işi olan işletme mantığına konsantre olabilir. Tek sorumluluk prensibi açışından bakıldığında da bu bir gerekliliktir.

Hollywood Prensibi

VIP (Very Important Person) olan şahıslara erişmek zordur. Onlar genelde "bizi aramayın, biz sizi ararız" şeklinde iletişimi tercih ederler. Holywood prensibi olarak bilinen bu prensibi IoC konseptini açıklamak için kullanabiliriz.

Bağımlılıkların enjekte edilmesi Hollywood prensibine göre çalışmaktadır. RentalController sınıfı kendi başına bir konstrüktör ya da fabrika metodu koşturarak bir service nesnesi edinmeye çalışmaz. Bunu yapsaydı eğer, o zaman bu VIP şahsı telefonda aramak ve benim service nesnesine ihtiyacım var demek gibi bir şey olurdu. Bunun yerine VIP şahıs, yani Spring RentalController sınıfında yer alan setService() metodunu kullanarak RentalController sınıfıyla iletişime geçmektedir. Spring RentalController sınıfına bir RentalService nesnesi enjekte edebilmek için setService() metodunu koşturmaktadır. Spring sınıfların set() metotlarını ya da konstrüktörlerini kullanarak gerek duyulan bağımlılıkları enjekte etmektedir. Bağımlılığı enjekte edebilmek için bu metotları koşturması, yani sınıfı araması gerekmektedir.

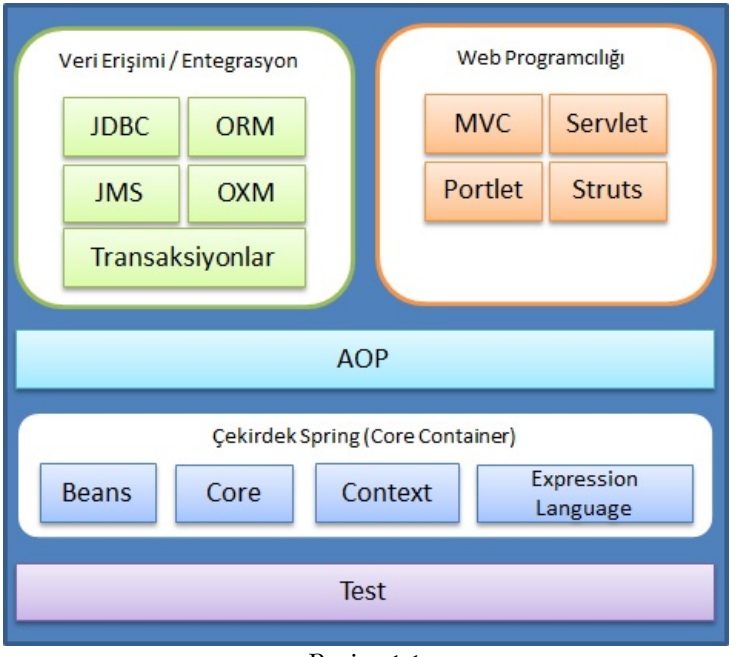
Spring koşturulacak metodun seçiminde konstrüktör ya da set() metoduyla sınırlı değildir. Hollywood prensibi sınıf bünyesinde yer alan sınıf metotları üzerinde de kullanılabilir. Bu Spring'in konfigürasyon dosyasında belirlenen herhangi bir sınıf metodunu koşturulabileceği anlamına gelmektedir. Kitabın on yedinci bölümünde inceleyeceğimiz Spring Task ve Scheduling modülünde herhangi bir POJO (Plain Old Java Object) sınıfın herhangi bir metodunu şu şekilde koşturmak mümkündür:

```
<task:scheduled ref="rentalDownloader"
        method="download" cron="*/5 * 9-17 * * MON-FRI"/>
```

Spring tarafından koşturulması gereken metodun ismi method element özelliğinde yer almaktadır. Görüldüğü gibi rentalDownloader nesnesi görevini yerine getirmek için hangi metodun koşturulması gerektiğini bilme sorumluluğundan arındırılmaktadır. Sadece konfigürasyon dosyası üzerinde değişiklik yaparak, POJO sınıfın çalışma tarzı adapte edilebilmektedir. Spring birçok modülünde bu mekanizmadan faydalanmaktadır.

Spring Modülleri

Spring değişik modüllerden oluşan modüler bir yapıya sahiptir. Spring bir kurumsal projeyi gerçekleştirmek için gerekli her şeyi ihtiva etmekle birlikte, değişik Spring modüllerini ihtiyaçlarımız doğrultuşunda seçerek, kullanabiliriz. Spring'in modüler yapısı genel hatlarıyla resim 1.1 de görülmektedir.



Resim 1.1

Spring 3.2.4 sürümü ile yirmi değişik modülden oluşmaktadır. Her modül kendi JAR dosyasında yer almaktadır. Bu modüllerde bazıları resim 1.2 de yer almaktadır.

Resim 1.2

Spring Modülleri İle Neler Yapabiliriz?

Bu bölümde altbaşlıklar halinde Spring modüllerini tanıtmak istiyorum.

Çekirdek Sunucu (Core Container) Modülü

Spring uygulamalarının temelini çekirdek (core) sunucu (container) oluşturmaktadır. Bir Spring uygulaması çalışmaya başladığında Spring tarafından yapılan ilk işlem, içinde Spring nesnelerinin (Spring Bean) yer aldığı ve bu nesneler arasında bağımlılıkların enjekte edilmesini sağlayan bir sunucu (container) oluşturmaktır. Bağımlılıkların enjekte edilmesi için BeanFactory kullanılır. BeanFactory yapısını kitabın üçüncü bölümünde inceleyeceğiz.

Çekirdek sunucu Core, Context, Beans ve Expression Language modüllerinden oluşmaktadır. Bu modüllerin ne olduklarını ve nasıl kullanılabileceklerini kitabın ikinci bölümünden itibaren örnek kodlar üzerinde inceleyeceğiz.

Diğer tüm Spring modülleri çekirdek sunucu üzerinde inşa edilmiş modüllerdir. Her Spring uygulaması mutlaka çekirdek sunucuyu oluşturan modüllerde yer alan sınıflar kullanılarak konfigüre edilir. Spring çekirdek sunucu yapı itibari ile bir EJB uygulama sunucusuna benzer. Bünyesinde yer alan tüm nesnelere konfigürasyonları doğrultusunda ihtiyaç duydukları servisleri sunar. Bunu yaparken AOP teknolojisini kullanır.

Spring AOP Modülü

Spring uygulamalarını POJO sınıfların oluşturduğunu söylemiştik. Bu sınıflar mevcut yapıları itibari ile bir kurumsal projelerin gereksinimlerini tatmin edecek yapıda değildirler. Örneğin EJB komponentler uygulama sunucusu tarafından güvenlik ya da otomatik transaksiyon yönetimi gibi yetilerle donatılırlar. Aynı şey Spring tarafından AOP kullanılarak POJO sınıflar üzerinde gerçekleştirilir. Örneğin sadece işletme mantığını ihtiva eden bir POJO sınıfa AOP kullanılarak transaksiyonel özellik kazandırılabilir. Kitabın dokuzuncu bölümünü AOP konusuna ayırdım.

Veri Erişimi Modülü

JDBC tabanlı veri tabanı işlemleri için Spring bünyesindeki JDBC modülünü kullanabiliriz. JDBC kodu yazmış olanlar bilirler. JDBC checked exception türünü kullandığı için JDBC kodu çok kısa zamanda okunmaz bir hale gelebilir. Spring bünyesinde yer alan JDBC modülü ile çok sade JDBC kodu yazmak mümkündür. Sağladığı DAO (Data Access Object) katmanı ile, veri katmanı ile veri tabanı arasında esnek bir bağın oluşmasını destekler. Kitabın beşinci

bölümü bu modülü tanıtmaktadır.

JDBC yerine Hibernate ya da EclipseLink gibi bir ORM (Object Relational Mapping) teknolojisini tercih edenler Spring ORM modülü ile bu teknolojileri Spring uygulamalarında kullanabilirler. Spring ORM modülü bir ORM çatısı olma iddiasını gütmemektedir. Daha ziyada mevcut ORM teknolojilerini entegre ederek, tek bir programlama modeli üzerinden kullanımlarını sağlamaktadır. Kitabın yedinci bölümünde Spring ile Hibernate, sekizinci bölümünde JPA (Java Persistence API) kullanımını yakından inceleyeceğiz.

Spring OXM (Object XML Mapping) modülü Java<=>XML dönüşümünü sağlamak için kullanabileceğimiz Spring modülüdür. Yine diğer modüllerde de olduğu gibi bir OXM çatısı olma iddiası gütmemektedir. Daha ziyada JAXB, Castor, XMLBEans ve XStream gibi teknolojileri kullanarak Java<=>XML dönüşümünü sağlamaktadır. Kitabın on ikinci bölümününde yer alan REST ve on dördüncü bölümününde yer alan Web Service konularında Spring OXM modülünün kullanımını inceleyeceğiz.

Spring JMS (Java Messaging Service) JMS mesajları oluşturmak (produce) ve tüketmek (consume) için kullanılan modüldür.

Transaksiyon modülü deklaratif ve programsal transaksiyon yönetimi yapılmasını sağlamaktadır. Kitabın altıncı bölümünde Spring ile transaksiyon yönetimini inceleyeceğiz.

Spring MVC Modülü

Spring genelde entegratif bir çatıdır, yani mevcut teknolojileri aynı çatı altında toplayarak, belli bir programlama modeli sunar. Bunun bozulduğu istisnalardan bir tanesi Spring MVC çatısıdır. Bu çatı ile web tabanlı uygulamalar geliştirmek mümkündür. Spring ile Struts ya da Wicket gibi web çatılarını kullanmak mümkün iken, Spring burada kendi web çatısını geliştirmeyi tercih etmiştir. Kitabın onuncu bölümünde Spring MVC web çatısını yakından inceleyeceğiz.

Spring Remoting Modülü

Spring MVC modülü ile web tabanlı uygulamalar geliştirmek mümkün iken, Spring Remoting modülü ile POJO bazlı sınıfları servis sunucusu haline dönüştürmek mümkündür. POJO sınıflar RMI (Remote Method Invocation), Hessian, Burlap, JAX-WS (Web Service) ve HTTP invoker protokol ve

teknolojileri kullanılarak servis sunucu haline getirilebilir. Ayrıca Spring Remoting ile mevcut servis sunucuları ile iletişimde kullanılabilecek kullanıcılar (client) oluşturulabilir. Kitabın on üçüncü bölümü bu modülün kullanılış tarzını tanıtmaktadır.

Spring Test Modülü

Yazılımcı olarak benim Spring'in en çok ilgimi çeken tarafı, test güdümlü yazılımı mümkün kılan bir test çatısına sahip olmasıdır. Spring Test JUnit ve TestNG çatılarını kullanarak birim ve entegrasyon testlerinin geliştirilmesini mümkün kılmaktadır. Kitabın on beşinci bölümünde Spring ile test seçeneklerini inceleyeceğiz.

Spring Uygulama Portföyü

Spring çekirdek uygulama çatısı haricinde, bu çekirdek çatı üzerine inşa edilmiş uygulamaları da ihtiva etmektedir. Bu uygulamalardan bazıları:

- ***Spring Integration*** - Spring programlama modelini kurumsal entegrasyon şablonları (Enterprise Integration Patterns) destekleyecek şekilde genişletir. Spring Integration birbirlerinden tamamen bağımsız olan yazılım modüllerinin mesajlaşma (messaging) teknikleri kullanılarak bir uygulama oluşturacak şekilde bir araya getirilmelerini amaçlamaktadır.
- ***Spring Batch*** - Kullanıcı arayüzüne ihtiyaç duymadan seri bir şekilde işlem yapmayı (batch application) mümkün kılar. Özellikle bankalar gibi müşteri işlemlerini mesai saatlerinden sonra topluca yapan kuruluşların bu yöndeki ihtiyaçlarını karşılamak amacıyla geliştirilmiştir.
- ***Spring Social*** - Spring uygulamalarını Facebook, Twitter, ve LinkedIn gibi sosyal medya platformları ile entegre etmek için geliştirilmiş uygulamadır.
- ***Spring Mobile*** - Spring MVC web yazılım çatısını genişleten ve mobil web uygulama yazılımını kolaylaştırmak için kullanılan uygulamadır.
- ***Spring Web Services*** - SOAP bazlı web servis uygulamaları geliştirmek için kullanılmaktadır.
- ***Spring Web Flow*** - Temelinde Spring MVC web çatısını kullanan Web Flow kurumsal bir işlemi birden fazla web sayfasına bölerek, işlemin adım, adım yapılmasını sağlayan uygulamadır. Web Flow sayfalar arası otomatik oturum ve durum yönetimini yapmaktadır.
- ***Spring LDAP*** - Spring bazlı uygulamaları için LDAP (Lightweight

Directory Access Protocol) kullanımını basitleştirmektedir.

- ***Spring Security*** - Spring uygulamalarında güvenlik konfigürasyonunu yapmak için kullanılan uygulama modülüdür.
- ***Spring Data*** - Relasyonel veri tabanı sistemleri yanı sıra map-reduce, NOSQL, bulut gibi yeni veri tabanı ve veri erişimi teknolojilerinin kullanımını kolaylaştırmaktadır. JPA, MongoDB, Neo4j, Redis, Hadoop, Gemfire, Rest, Solr, CouchBase ve ElasticSearch gibi teknolojileri destekleyen alt modülleri mevcuttur.
- ***Spring XD*** - Büyük veri (big data) uygulamalarında import, export, analiz ve batch işleme gibi işlemleri kolaylaştırmak için oluşturulmuş uygulamadır.
- ***Spring Roo*** - Java ile uygulama geliştiren yazılımcılar için hızlı uygulama geliştirme (rapid application development) aracıdır.

Spring 3 İle Gelen Yenilikler

Bu bölümde Spring 3.0, 3.1 ve 3.2 sürümlerinde yer alan yenilikleri sizlerle paylaşmak istiyorum. Bu sürümlerde göze çarpan yenilikler şunlardır:

Spring 3.0

- Bu sürüm ile tüm Spring çatısı Java 5 ile gelen Generics, Varargs ve diğer Java dili yeniliklerini kullanacak şekilde elden geçirilmiştir. Spring'de anotasyon desteği 2.5 sürümü ile gelmiş olsa bile, Spring 3.0 sürümü ile bu destek daha da artırılmış ve JSR-330 ile gelen standart Java anotasyonların kullanımı mümkün hale gelmiştir. Bu şekilde anotasyon bazlı konfigürasyon kullanıldığında standart Java anotasyonları kullanılarak, kod bazındaki Spring çatısına olan bağımlılık ortadan kaldırılabilmektedir.
- Spring uygulamalarında konfigürasyon XML dosyaları üzerinden yapılmaktadır. Spring 3.0 sürümü ile XML dosyası kullanmadan anotasyon bazlı konfigürasyon yapmak mümkün hale gelmiştir.
- Spring 3.0 konfigürasyon imkanlarını daha esnek hale getirmek için Spring Expression Language (SpEL) modülünü ihtiva etmektedir.
- 3.0 sürümü ile Spring MVC uygulamalarını daha kolay konfigüre etmek için yeni XML mvc isim alanı oluşturulmuştur. Yeni eklenen @CookieValue ve @RequestHeaders gibi anotasyonlarla Spring MVC uygulamalarının anotasyon bazlı konfigürasyonu genişletilmiştir.
- Web uygulamaları geliştirmek için kullanılan Spring MVC, 3.0 sürümü ile

REST (Representational State Transfer) desteği sağlamaktadır. Spring MVC ile bir REST uygulaması oluşturmak için Spring MVC controller sınıfları kullanılmaktadır. RestTemplate kullanıcı (client) uygulamalar geliştirmek için kullanılmaktadır.

- jdbc isim alanında yer alan embedded-database konfigürasyon elementi ile HSQL, H2, ve Derby gibi veri tabanı sistemlerinin kullanımı kolaylaştırılmıştır.
- Java EE 6 ile kullanıma sunulan @Asynchronous anotasyonu ile metotlar asenkron koşturulabilmektedir. Spring 3.0 sürümünde yer alan @Async anotasyonu ile bu desteği sağlamaktadır.
- Spring 3.0 JSR-303 (Bean Validation) bünyesinde yer alan anotasyonları desteklemektedir.

Spring 3.1

- Yeni bir caching modülü (Cache Abstraction) ihtiva etmektedir.
- Bu sürümle Spring bean tanımlamalarını profil bazında gruplamak (bean definition profiles) mümkün hale gelmiştir. Profiller yardımı ile uygulama değişik ortamlara göre adapte edilmiş konfigürasyon dosyalarını kullanabilmektedir.
- Oluşturulan yeni Environment isimli sınıf ile profil bazlı bilgilerin yer aldığı yeni bir alan oluşturulmuştur. Bu alan içinde profil bilgileri yanı sıra tanımlanan değişken (property) değerleri de yer almaktadır. Environment sınıfı kullanılarak bu bilgilere ulaşılabilir.
- constructor-arg elementini daha kısa yazmak için c isim alanı oluşturulmuştur. Bunun kullanımını üçüncü bölümde yakından inceleyeceğiz.
- Bu sürüm Hibernate 4.x desteği vermektedir.
- 3.1 sürümü öncesi enjeksiyon için kullanılan set metotlarının void veri tipinde bir değeri geri vermeleri gerekiyordu. Bu yeni sürümle set metotları herhangi bir yapıda olabilmektedir.
- Spring'in test çatısı olan Spring TestContext bünyesindeki @ContextConfiguration anotasyonu @Configuration anotasyonunu taşıyan konfigürasyon sınıflarını desteklemektedir. Ayrıca entegrasyon testlerinde kullanılmak üzere değişik uygulama profillerini destekleyen @ActiveProfiles anotasyonu oluşturulmuştur.
- JPA bünyesinde sınıflar META-INF/persistence.xml dosyasında tanımlanmaktadır. Spring 3.1 sürümü ile gelen

LocalContainerEntityManagerFactoryBean ile classpath içinde yer alan sınıflar otomatik olarak taranarak persistence.xml kullanmayan bir JPA altyapısı oluşturulabilmektedir.

- Spring MVC controller sınıflarında kullanılan @RequestMapping anotasyon tanımlaması consumes ve produces elementleri kullanılarak genişletilmiştir. consumes controller sınıfının hangi türde verileri işleyebileceğini belirlerken, produces kullanıcıya gönderilecek cevabın hangi formatta olması gerektiğini tanımlamaktadır. Böylece örneğin bilgileri XML formatında alan ve kullanıcıya cevabı JSON formatında gönderen controller sınıfları tanımlamak mümkün hale gelmiştir.
- Yeni oluşturulan @RedirectAttributes anotasyonu ile controller metotlarında yönlendirme (redirect) işlemi için parametre tanımlaması yapılabilmektedir.
- @RequestBody anotasyonu kullanılan bir controller metodunda @Valid anotasyonu kullanılarak otomatik validasyon işlemi yapmak mümkün hale gelmiştir.

Spring 3.2

- Spring MVC uygulamalarını uygulama sunucusuna bağımlı olmadan test edebilmek için yeni Spring MVC Test çatısı oluşturulmuştur.
- Bu sürüm Java EE 7'nin bir parçası olan JCache desteği sağlamaktadır.
- Spring MVC, Servlet 3 sürümünde tanımlanan asenkron metot koşturma (asynchronous request processing) özelliğini desteklemektedir.
- RestTemplate HTTP cevaplarında (response) yer alan verileri Java Generics kullanarak (örnegin List) edinebilmektedir.
- Spring bu sürümünde Jackson JSON 2 kütüphanesini ve bir şablon (template) yönetim çatısı olan Tiles 3 sürümünü desteklemektedir.

Spring'in Uygulama Geliştirmedeki Rolü

Spring kurumsal Java projeleri geliştirmek için geniş çaplı altyapısal destek sağlamaktadır. Entegratif yönüyle mevcut Java API (Application Programming Interface) ve çatıların (framework) kullanımını mümkün olduğu kadar tek bir programlama modelinde toplamaktadır. Değişik API ve çatıları tek bir programlama modeli ile kullanabilmek programcıların verimliliğini artıran bir durumdur.

Spring plumbing code olarak isimlendirilen, işletme mantığının mecbur kılınan programlama modeli neticesinde gereksiz kod kalabalığı ile sişmesini sunduğu konfigürayon yöntemleri ile engellemektedir. Böylece POJO sınıflar geliştirerek, sadece işletme mantığına konsantre olmak mümkün hale gelmektedir.

Spring'in çekirdeği uygulama konfigürasyonu, kurumsal entegrasyon, test etme ve veri erişimi konuları için çözümler sunmaktadır. Spring ile bir uygulamayı değişik modülleri bir araya getirerek oluşturmak mümkündür. Modüllerin birbirlerini bulmaları gerekliliği yoktur. Her modül Spring konfigürasyonu aracılığı ile uygulamanın genel yapısına zarar vermeden başka bir modül ile yer değiştirebilir. Uygulamayı oluşturan modüllerin sessiz, sedasız değiştirilebilir yapıda olması, uygulamanın test edilebilirliğini olumlu etkilemektedir. Sunduğu test imkanları ile Spring uygulamalarını, buna Spring MVC ile oluşturulan web uygulamaları da dahildir, uygulama sunucusu olmadan test etmek mümkündür.

Kaynakları yönetmesi, veri erişimi için kullanılan API'lerin kullanımını kolaylaştıran sınıflar sunması, JDBC, Hibernate, JPA ve IBatis gibi popüler veri erişimi teknolojilerini desteklemesi ile Spring veri erişimi programcılığını kolaylaştırmaktadır.

Spring Struts, Wicket ya da JSF gibi web çatıları ile entegre edilebilmektedir. Bu entegrasyon ile bu çatılarda Spring'in sunduğu uygulama konfigürasyonu modeli kullanılabilmektedir. Bunun yani sıra ihtiva ettiği Spring MVC ve Spring Web Flow çatıları ile web uygulama geliştirmeyi desteklemektedir.

Spring Yazılım Geliştirme Ortamı

Spring STS (SpringSource ToolSuite) ismini taşıyan ve Eclipse bazlı bir yazılım geliştirme ortamına sahiptir.

STS ile tam teşekküllü bir yazılım geliştirme ortamı edinmenin yanı sıra mevcut bir Eclipse Helios (3.6), Indigo (3.7), Juno (3.8/4.2) ya da Keppler 4.3 sürümü Eclipse Marketplace üzerinde STS plugin seti yüklenerek Spring yazılım ortamına dönüştürülebilir.

Resim 1.3

Spring Jar Dosyalarını Nasıl Edinebilirim?

Spring çatısını oluşturan Jar dosyalarını edinmenin en hızlı yolu bir Maven projesi oluşturmak ve aşağıda yer alan Maven bağımlılığını projeye eklemektir. Kitabın her bölümü için oluşturduğum Maven projelerinde bu şekilde gerekli Jar dosyalarını projeye ekledim.

```
Kod 1.1 - pom.xml

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.5.RELEASE</version>
    </dependency>
</dependencies>
```

Eğer Maven kullanmıyorsanız, Spring.io sayfasından güncel Spring sürümünü

edinebilirsiniz.

Spring Hello World

Her teknolojiyi öğrenmeye klasik Hello World uygulaması ile başlanır. Bu bölümde Spring ile bu tarz bir uygulamanın nasıl geliştirilebileceğini bir örnek üzerinde göstermek istiyorum.

İlk işlem olarak HelloWorldService isminde bir interface sınıf tanımlıyoruz. Bu interface sınıf kod 1.2 de yer almaktadır. getMessage() metodu istediğimiz türde bir mesajı geri verecektir.

```
Kod 1.2 - HelloWorldService

public interface HelloWorldService {
    String getMessage();
}
```

HelloWorldService sınıfını implemente eden sınıf kod 1.3 de yer almaktadır. getMessage() metodunu Hello World kelimelerini geriye verecek şekilde yapılandırıyoruz.

```
Kod 1.3 - HelloWorldServiceImpl

public class HelloWorldServiceImpl implements HelloWorldService {
    @Override
    public String getMessage() {
        return "Hello World";
    }
}
```

Şimdi HelloWorldService sınıfını kullanan başka bir sınıf tanımlayalım. Kod 1.4 de yer alan MessageManager sınıfı bünyesinde service ismi altında HelloWorldService sınıfını kullanmaktadır. Bir Spring anotasyonu olan @Autowired ile MessageManager sınıfına, daha doğrusu bu sınıftan olan nesneye bir HelloWorldService nesnesi enjekte edilmektedir.

```
Kod 1.4 - MessageManager

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
```

```
@Component
public class MessageManager {

    @Autowired
    HelloWorldService service;

    public void printMessage() {
        System.out.println(this.service.getMessage());
    }
}
```

Spring uygulamasını konfigüre etmek ve koşturmak için Application sınıfını (kod 1.5) oluşturuyoruz. @Configuration sınıf bazlı Spring konfigürasyonu kullandığımıza işaret etmektedir. Bu Spring uygulaması sıfır XML konfigürasyonu kullanmaktadır. @ComponentScan ile Spring'e gerekli sınıfları classpath içinde araması gerektiğini belirtiyoruz.

@Bean anotasyonu bir Spring bean tanımlamak için kullanılmaktadır. Kod 1.4 de yer alan MessageManager sınıfına HelloWorldService tipinde bir nesne enjekte edebilmek için Spring'in hangi HelloWorldService implementasyon sınıfının kullanıldığını bilmesi gerekmektedir. Bu uygulamada kullandığımız HelloWorldService implementasyonu kod 1.3 de yer alan HelloWorldServiceImpl sınıfıdır. Spring getMessageService() metodunu yeni bir HelloWorldServiceImpl nesnesi oluşturmak için kullanacak, akabinde bu nesneyi kod 1.4 de yer alan service değişkenine enjekte edecektir, çünkü bu değişken @Autowired ile işaretlenmiştir.

```
Kod 1.5 - Application

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.
                AnnotationConfigApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class Application {

    @Bean
    public HelloWorldService getMessageService() {
```

```
        return new HelloWorldServiceImpl();
    }

    public static void main(final String[] args) {
        final ApplicationContext context =
             new AnnotationConfigApplicationContext(
                               Application.class);
        final MessageManager manager =
                context.getBean(MessageManager.class);
        manager.printMessage();
    }
}
```

Kod 1.5 de yer alan Application sınıfı hem uygulamayı konfigüre etmek, hem de koşturmak için kullanılmaktadır. @Configuration, @ComponentScan ve @Bean anotasyonları uygulamayı konfigüre etmek için kullanılırken, main() metodu uygulamayı koşturmaktadır. main() metodu bünyesinde yeni bir AnnotationConfigApplicationContext nesnesi oluşturulmaktadır. Bu nesne konfigüre ettiğimiz Spring uygulamasını temsil etmektedir. Bu nesne üzerinden getBean() metodu aracılığı ile bir MessageManager nesnesi edinebiliriz. manager.printMessage() metodu koşturulduğunda ekranda Hello World kelimeleri yer alacaktır.

Bu Spring ile yaptığımız çok basit bir dependency injection örneğiydi. Şimdi sır perdesini aralamaya ve Spring'i daha yakından tanımaya hazır mısınız? Öyleyse Spring trenine binip, on sekiz durağa (kitabın bölümleri) uğrayarak, Spring ile neler yapabileceğimizi birlikte görelim. Birlikte yapacağımız bu yolculukta Spring hakkında umduklarınızı bulacağınızı ümit ediyorum.

Hazırsanız tren kalkıyor...